НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2018р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки ”

на тему: «Інформаційна масштабована система обробки даних про бронювання (залізничних) квитків в реальному часі на основі нереляційної розподіленої бази даних»

Виконала: студентка IV курсу, групи ТР-51

Стрелецька Анастасія Максимівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доц. Шпурик В.В _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2018р.

ЗАВДАННЯ
на дипломну роботу студенту
Стрелецької Анастасії Макисмівни
(прізвище, ім’я, по батькові)

1. Тема роботи: «Інформаційна масштабована система обробки даних про бронювання (залізничних) квитків в реальному часі на основі нереляційної розподіленої бази даних»
керівник роботи Шпурик Вадим Вадимович ,кандидат технічних наук, доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мова програмування Java, середовище IntelliJ IDEA, розподілена нереляційна база даних MongoDB_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити масштабовану інформаційну систему обробки даних про бронювання залізничних квитків з використанням нереляційної розподіленої бази даних, як ефективного рішення рівня бази даних.
5. Перелік ілюстративного матеріалу архітектура системи, графічне представлення інтерфейсу, приклади роботи програмного модулю_____
6. Дата видачі завдання ” 10 ” жовтня 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	14.10.2018-23.12.2018	
2.	Розробка архітектури та загальної структури системи	2.02.2019-3.03.2019	
3.	Розробка структур окремих підсистем	4.03.2019-14.04.2019	
4.	Підготовка матеріалів	15.04.2019-18.04.2019	
5.	Програмна реалізація системи	18.04.2019-14.05.2019	
6.	Захист програмного продукту	15.05.2019	
7.	Оформлення пояснювальної записки	16.05.2019-3.06.2019	
8.	Передзахист	28.05.2019	
9.	Захист	17.06.2019-22.06.2019	

Студент _____ Стрелецька А. М.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Шпурик В. В.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Пояснювальна записка містить 54 сторінок, включає 35 рисунків, 2 таблиці, 7 посилань.

Метою дипломної роботи є створення інформаційної системи бронювання (залізничних) квитків. Створено клієнтський додаток за допомогою мови програмування Java в середовищі розробки IntelliJ IDEA. Додаток працює з нереляційною розподіленою базою MongoDB.

Розроблено програмний продукт бронювання (залізничних) квитків з функціями бронювання квитків з вибором місця для пасажира, авторизації в системі, з наданням різних рівнів доступу, відміною бронювання квитків, бронювання квитків, перегляд вільних для бронювання квитків, користувачів системи, забронюваних квитків, реєстрацією.

Ключові слова: масштабована система, інформаційна система, обробка даних, залізничні квитки, нереляційна база даних, розподілена база даних, бронювання квитків, великі дані, Java, .

ABSTRACT

The explanatory note contains 54 pages, including 35 illustrations, 2 tables, links.

The purpose of the thesis is to create an information system for reservation (rail) tickets. The client application was created using the Java programming language in the IntelliJ IDEA development environment. The application works with a non-relational distributed MongoDB database.

A software product was developed for booking (rail) tickets with functions provided by choosing a place for a passenger, authorizing the system, providing different levels of access, refusing to book tickets, booking tickets, viewing free tickets, user accounts.

Keywords: scalable system, information system, data processing, railway tickets, non-relational database, distributed database, reservation of tickets, large data, Java.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ТЕРМІНІВ

HTML	Hypertext Markup Language — Мова розмітки гіпертекстових документів
XML	Extensible Markup Language — розширювана мова розмітки
DTO	Data Transfer Object — шаблон проектування, для передачі даних між підсистемами програми
DAO	Data Access Object — Об’єкт доступу до даних
SQL	Structured Query Language — Стандартна мова для керування реляційними базами даних
NoSQL	Non SQL— технологія зберігання нереляційних баз даних
JSP	JavaServlet Page — технологія для створення динамічних HTML-сторінок
JSON	JavaScript Object Nonation— файл, який має простий формат для обміну даними
POM	Project Object Model — це XML-файл, який зберігає всю інформацію про проект та налаштування
ACID	Atomicity, Consistency, Isolation, and Durability — це концепція, що посилається на чотири властивості транзакцій системи баз даних: атомність, послідовність, ізоляція та довговічність
BASE	Basically Available, Soft state, Eventual consistency — акронім, який використовується для опису властивостей певних баз даних, як правило, баз даних NoSQL.
Sharding	Шардінг — це метод розподілу даних між декількома машинами для підтримки розгортання з великими наборами даних і операціями високого рівня пропускної здатності.

ВСТУП

Одним з важливих етапів, при розробці програмного продукту, являється розробка архітектури рівня бази даних. Питання щодо ефективного зберігання та використовування великих даних, та ефективного вилучення інформації, стає дуже важливим у багатьох галузях нашого часу. Щоб підтримати це, промисловість почала рухатися до нових систем, які називаються нереляційними або NoSQL.

NoSQL змінює правила різними способами, і використання бази даних NoSQL найкраще супроводжується відповідною зміною архітектури додатків.

Рух NoSQL продовжує набирати обертів, так як розробники все більше втомлюються від традиційного управління базами даних на основі SQL і шукають поліпшення в технології зберігання даних.

Підхід до бази даних NoSQL характеризується відмовою від складності серверів на основі SQL. Логіка перевірки, управління доступом, зіставлення запитуваних індексованих даних, кореляції пов'язаних даних, вирішення конфліктів і підтримки обмежень цілісності переміщується з рівня бази даних. Це дозволяє NoSQL зосередитися на винятковій продуктивності і масштабованості. Фундаментальні ж проблеми, які пов'язані з даними, не зникають, а повинні перейти до програмного рівня, чим і характеризується робота з нереляційною моделлю баз даних NoSQL.

Особливістю дипломної роботи є використання саме нереляційної розподіленої бази даних, як одного з ефективних архітектурних рішень рівня бази даних для роботи з великими даними, використовуючи реалізацію інформаційної системи про бронювання (залізничних) квитків.

1. АКТУАЛЬНІСТЬ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ВИБОРУ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

Сьогодні сучасні підприємства думають про кращі способи зберігання та керування своїми даними - будь то для кращого розуміння клієнта, адаптації до зміни очікувань користувачів, або для того, щоб подолати конкурентів на ринку з новими додатками та бізнес-моделями. Як наслідок, багато припущень, які призвели до розробки попередніх реляційних баз даних, змінилися:

- Вимоги до більш високої продуктивності розробників і більш швидкого виходу на ринок, з традиційними жорсткими реляційними моделями даних і розробкою водоспаду монолітних додатків, що дають можливість гнучкому методології, мікросервісам і DevOps, стискаючи цикли випуску з місяців і років на дні і тижні.
- Необхідність управління масовим збільшенням нових, швидко змінюваних типів даних - структурованих, напівструктурованих і поліморфних даних, що генеруються новими класами веб-, мобільних, соціальних і IoT-додатків.
- Перехід до розподілених систем і хмарних обчислень, що дозволяє розробникам використовувати інфраструктуру обчислень і сховищ на вимогу, з можливістю обслуговувати аудиторію в будь-якому місці, де вони працюють і грають по всьому світу. за суверенітет даних.
- Як наслідок, з'явилися нереляційні бази даних, такі як MongoDB, з метою задоволення вимог нових додатків і модернізації існуючих навантажень.

1.1 Аналіз моделей архітектури рівня бази даних

Поява нової документно-орієнтованої нереляційної бази даних MongoDB надала можливість широкого вибору розробки моделі архітектурного рівня бази

даних, яка почала конкурувати з, вже набувшими популярності та високого рейтингу в користуванні, реляційними базами даних, такими як MySQL.

1.1.1 Збереження даних в базах даних

Зберігання даних відбувається досить різними підходами. Коли MySQL, як і інші реляційні системи ,зберігає дані в таблицях і використовує структурований мова запитів (SQL) для доступу до бази даних, MongoDB зберігає дані як документи у двійковому поданні BSON (Binary JSON). В MongoDB пов'язана інформація зберігається разом для швидкого доступу до запитів за допомогою мови запитів MongoDB. Поля можуть варіюватися від документа до документа; немає необхідності декларувати структуру документів у системі - документи самоописуються, тоді як MySQL ви заздалегідь визначаєте схему бази даних на основі ваших вимог і встановлюєте правила для регулювання відносин між полями у ваших таблицях.

Проблеми в MySQL виникають при необхідності зміни схеми бази даних. Необхідність в міграції, може призвести до втрати продуктивності обробки даних. Тоді як в MongoDB додавання нового поля може бути створено без урахування всіх інших документів у збірці, не оновлюючи центральний системний каталог і не виводячи систему в автономний режим. Це рішення NoSQL зявилось з впровадженням авто-шардінгу, і вбудованої реплікації для кращої масштабованості і високої доступності.

1.1.2 Модель цілісності - ACID і BASE

ACID - це концепція, що посилається на чотири властивості транзакцій системи баз даних: атомність, послідовність, ізоляція та довговічність.

Абревіатура BASE означає:

- Basically Available(англ. в основному доступний): Система гарантовано доступна для запитів усіма користувачами.

- Soft State (англ. м'який стан): Значення, збережені в системі, можуть змінюватися через можливу модель узгодженості, як описано в наступному маркері.
- Eventually Consistent(англ. врешті-решт послідовний): Оскільки дані додаються до системи, стан системи поступово реплікується по всіх вузлах.

База даних MongoDB слідує моделі BASE. Вона підтримує автономні оновлення на рівні одного документа. Іншими словами, якщо відбувається оновлення двох значень в документі, всі ці два значення успішно оновлюються або залишаються незмінними (рисунок 1.1)

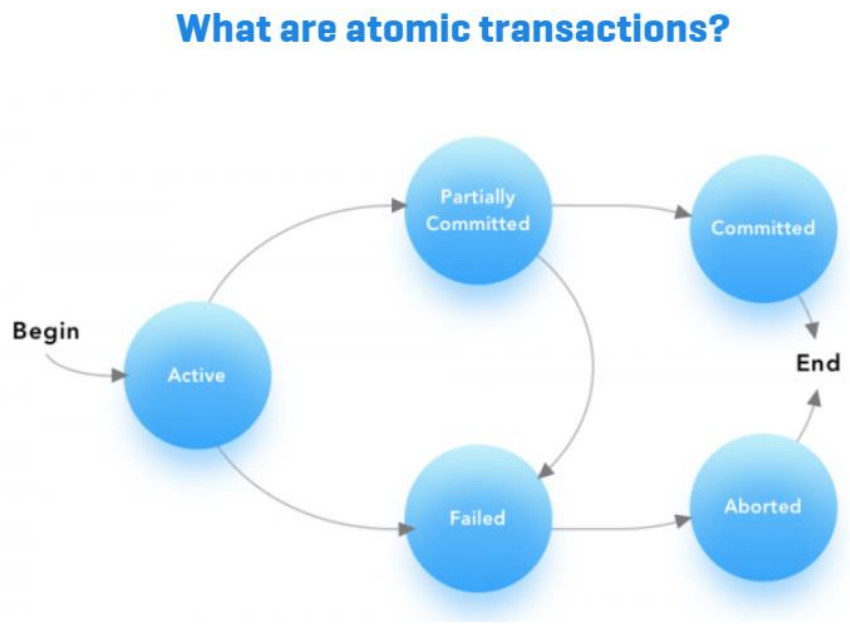

Рисунок 1.1 — Робота автономності оновлення

Дотепер MongoDB не підтримувала транзакцій і не гарантувала узгодженість даних. Але оновлення MongoDB 3.6, такі як введення клієнтських сесій та випадкова послідовність, призвело до надання більшої гнучкості розробнику задля точних налаштувань бажаної послідовності, яку вимагає програма.

MongoDB забезпечує налаштовувану модель узгодженості через параметри readConcern (дозволяє контролювати узгодженість і ізоляційні властивості даних, що зчитуються з наборів реплік і реплік набору осколків.) і writeConcern (описує рівень

підтвердження від MongoDB для операцій запису до автономного набору реплік або кластерів.). Причинно-узгоджений сеанс означає, що пов'язана послідовність операцій зчитання і запису має причинно-наслідковий зв'язок, що відображається їх впорядкованістю. Програми повинні переконатися, що тільки один потік за раз виконує ці операції в сеансі клієнта.

MySQL використовую наступну модель ACID (атомна, послідовна, ізольована і міцна). Це означає, що як тільки транзакція завершена, дані залишаються послідовними і стабільними на диску.

Це цілком підходить для програм, які не можуть нести втрати даних або невідповідності. Однак транзакції відомі для обмеження горизонтальної масштабованості.

1.1.3 Розподіл даних методикою Sharding

Sharding - це метод розподілу даних між декількома машинами для підтримки розгортання з великими наборами даних і операціями високого рівня пропускної здатності.

Високі швидкості запитів і набори даних, більші, ніж системна оперативна пам'ять, можуть надавати навантаження на можливості вводу-виводу дискових накопичувачів і процесора сервера. Ці проблеми вирішуються шляхом горизонтального і вертикального масштабування.

- Вертикальне масштабування передбачає збільшення ємності одного сервера шляхом додавання додаткової оперативної пам'яті, потужного процесора або місця для зберігання.
- Горизонтальне масштабування включає поділ набору даних і завантаження на кілька додаткових серверів. Кожна машина обробляє частину робочого навантаження з порівняно меншою вартістю, ніж високотехнологічне обладнання для однієї машини.

Sharding MongoDB має можливість розбивати колекцію на підмножини даних, щоб зберігати їх на декілька частин. Це дозволяє програмі виходити за межі ресурсів

автономного сервера або набору реплік. Крім того, він може обробляти розподіл даних через будь-яку кількість вузлів, щоб максимізувати використання дискового простору та динамічно запитувати баланс навантаження. Вона також надає користувачам можливість автоматизованого збою та надмірності.

Як і MySQL, MongoDB sharding має можливість виконувати розподіл даних на основі діапазону. MongoDB також підтримує автоматичне розподіл обсягу даних і прозору маршрутизацію запитів.

Кластер MongoDB складається з shard і, mongos (маршрутизатор запитів) і конфігураційних серверів (зберігання метаданих та конфігураційних налаштувань).

Монго дуже ефективний в деяких областях, де реляційні бази даних є особливо слабкими, як масштабування невідомих додатків, автоматичного розгортання.

Хоча shanding в MongoDB стандартизований, архітектори баз даних повинні мати на увазі наступні міркування:

Carnality - Виберіть клавішу shard, яку можна легко розділити пізніше, якщо розмір бази даних перевищує розмір шматка.

- Чутливість — вибір ключа shard, який можна розділити пізніше, якщо відбувається перевищення розміру його частини.
- Розподіл — ключ shard повинен поширюватися в рівномірному розподілі, щоб уникнути незбалансованості.
- Запит — кожен із запитів призведе до одиничного shard ключа, якщо будь-який з запитів має ключ shard. В іншому випадку він генерує запити для кожного shard.

В певний момент часу необхідність масштабування системи структури бази даних MySQL. Буде обрано рішення реплікації даних. Хоч і до MySQL досить легко додати шкалу читання, але яким буде масштаб даного запиту? Тут і відбувається складний етап.

При розробці Master-Master реплікації, як зображено на рисунку 1.2, дані успішно реплікуються при зміні в будь-якому з серверів.

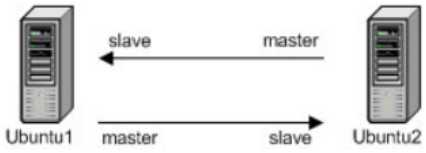

Рисунок 1.2 — Master-master реплікація в MySQL

При Multi-master реплікації виникають наступні проблеми(рисунок 1.3):

- нестійка робота автоінкременту;
- читання з відсталого сервера може мати застарілі результати;
- реплікація кільця досить складна в разі виникнення помилок;
- однопоточна реплікація, схильна до відставання.

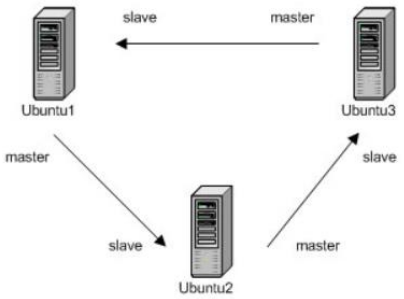

Рисунок 1.3 — Multi-master реплікація в MySQL

На відміну від MongoDB, MySQL не існує стандартної реалізації sharding. Хоча MySQL пропонує два шляхи для sharding тобто: кластер MySQL - вбудована функція автоматичного керування функціями і MySQL Fabric - офіційна структура sharding, але вони рідко розгортаються. Звичайна практика полягає в тому, аби підключити власний фреймворк шардингу, як і зробили Facebook та Pinterest.

В MongoDB присутня підтримка реплікації master-slave (рисунок 1.4). Використовується набори реплік для створення декількох копій даних. Кожному вузлу набору реплік буде призначена первинна або вторинна роль у будь-якій точці процесу. Читання / запис виконується на первинній копії, а потім реплікується на вторинні репліки(вузли).

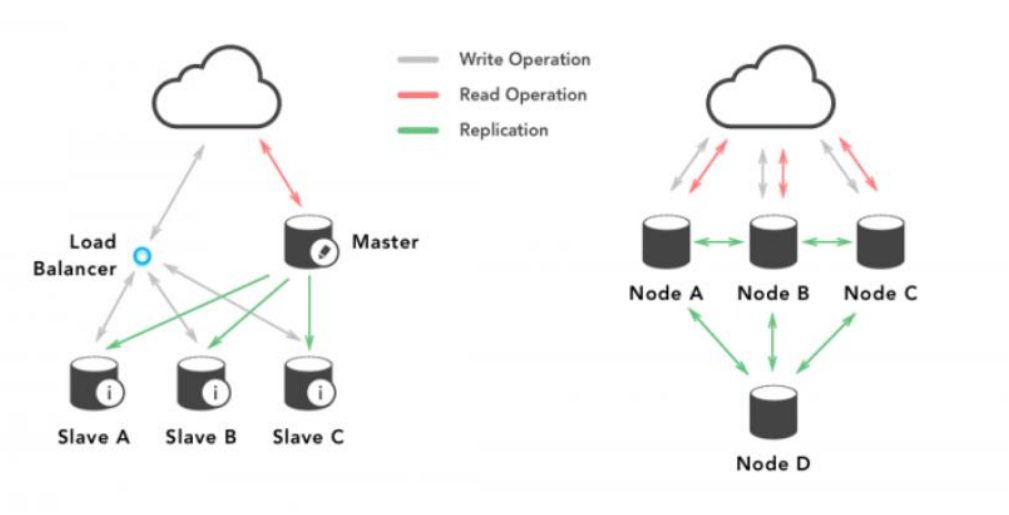

Рисунок 1.4 — Master-slave реплікація та master-master реплікація відповідно

MySQL підтримує як реплікацію master-slave, так і реплікацію master-master. Реплікація з декількома джерелами дає можливість паралельно копіювати дані з декількох майстрів. У реплікації master-slave узгодженість не занадто складна, оскільки кожен фрагмент даних має рівно одного власника. При реплікації master-master MySQL пропонує всі модливості задля обробки даних без жодної точки відмови.

1.1.4 Продуктивність обробки даних різних баз даних

Люди бачать реальну продуктивність MongoDB в основному тому, що MongoDB дозволяє робити запити іншим способом, який є більш розумним для робочого навантаження.

Наприклад, розглянемо конструкцію, яка зберігає багато інформації про складний об'єкт у нормалізованому вигляді. Для цього можливе використання двадцяти таблиць в MySQL (або будь-який реляційний базі даних) для зберігання даних у звичайній формі, з багатьма індексами, необхідними для забезпечення реляційної цілісності між таблицями.

Тепер розглянемо той самий дизайн з пакетом даних. Якщо всі ці пов'язані таблиці підпорядковані головній таблиці (а вони часто є), то буде можливість

змоделювати дані таким чином, що весь об'єкт буде зберігається в одному документі. У MongoDB можна зберігати дані як єдиний документ, в єдиній колекції. Тут починається MongoDB, що забезпечує чудову продуктивність.

У MongoDB, щоб отримати всю сутність, потрібно виконати:

- один пошук індексу на колекції ;
- отримати вміст однієї сторінки бази даних (фактичний бінарний документ json)

У MySQL з 20 таблицями потрібно виконати:

- один пошук індексу на кореневій таблиці;
- з кластерним індексом можна припустити, що значення кореневого рядка знаходяться в індексі;
- 20 + діапазонів пошуку ;

Загальна сума для mysql, навіть припускаючи, що всі індекси знаходяться в пам'яті (що важче, тому що їх у 20 разів більше), становить близько 20 пошуків діапазону.

Таким чином, для цього прикладу, остаточний показник становить приблизно в 20 разів більше з MySQL, порівняно з MongoDB.

Таким чином, MongoDB може підвищити продуктивність у деяких випадках використання.

Обходячи приклад, описаний вище, підвищення продуктивності можна здобути, розробивши архутектурний рівень бази даних з використанням нереляційної розподіленої бази даних.

Розробка системи з горизонтальним масштабуванням, забезпечить систему про про бронювання залізничних квитків підвищенням продуктивності обробки даних.

Висновки до розділу:

В даному розділі було описано та проаналізовано актуальність рішення розробки масштабованої системи на основі розподіленої нереляційної бази даних.

2. ЗАДАЧА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ МАСШТАБОВАНОЇ СИСТЕМИ ОБРОБКИ ДАНИХ ПРО БРОНЮВАННЯ (ЗАЛІЗНИЧНИХ) КВИТКІВ В РЕАЛЬНОМУ ЧАСІ НА ОСНОВІ НЕРЕЛЯЦІЙНОЇ РОЗПОДІЛЕНОЇ БАЗИ ДАНИХ

Перед виконанням дипломної роботи підприємством була поставлена задача організувати архітектурний рівень бази даних для ефективної обробки великих даних, використовуючи нереляційну базу даних. Обрано інформаційну систему бронювання залізничних квитків для впровадження архітектурного рівня, з можливістю масштабування сервісу.

Задача покращення архітектурного рівня баз даних необхідна для логічної структури збереження даних, швидкості обробки даних. Використовуючи сучасні програмні технології, розробити систему з використанням нереляційної бази даних на основі горизонтального розподілення даних.

Розроблена система повинна мати наступні критерії:

- масштабування системи при необхідності збільшення ресурсів обробки даних ;
- швидка обробка запитів, адже саме розподілення системи допомагає працювати з удосконаленням її продуктивності;
- клієнт-серверний шаблон ПО.

В дипломній роботі буде продемонстровано використання власного програмного продукту для автоматизації бронювання залізничних квитків, а також сучасної архітектурної моделі рівня бази даних для обробки великих обсягів даних.

Програмний продукт повинен мати такі функції:

- авторизація в системі;
- перегляд (залізничних) квитків;
- перегляд користувачів системи;

- можливість забронювати залізничних квитків;
- можливість надавати різні рівні доступу до системи.

Висновки до розділу:

У розділі описано мету дипломної роботи та оголошені вимоги до розробки системи та розробленого додатка в цілому.

3. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ

Важливим завданням у розробці програмного продукту є вибір технологій, які були використані для розробки інформаційної системи про бронювання залізничних квитків.

Для створення даної системи було використано інструменти, які зображенні на рисунку 3.1.

Рисунок 3.1 — Інструменти реалізації програмного забезпечення

Для реалізації інформаційної системи про бронювання залізничних квитків було використано:

- середовище розробки IntelliJ IDEA;
- мову програмування Java для написання серверної частини;
- мову для розмітки гіпертексту HTML з використанням технології JSP для організації веб-інтерфейсу;

- мову розмітки CSS з використання бібліотеки Bootstrap для розробки графічного інтерфейсу користувача;
- технологію для забезпечення автоматизації збірки проекту Maven;
- Apache Tomcat для розгортання застосунку на локальному сервері;
- Git для версіювання розробленої системи;
- базу даних MongoDB;
- реляційну базу даних MySQL для обґрунтування та порівняння з архітектурним рівнем, який реалізован з використанням нереляційною, розподіленої бази даних;
- Junit, призначений для написання та запуску тестів мовою програмування Java, та PowerMock, як генератор mock об'єктів під час виконання тестування.

Аби інформаційна система про бронювання (залізничних) квитків мала можливість не залежати від пристроя, на якому буде здійснюватись вхід у систему, тому були обрані незалежні від операційної системи технології — Java, HTML, Maven та MongoDB.

Мовою програмування високого рівня була обрана Java, не лише через зазначену кроссплатформеність, а тому що вона має багато бібліотек, які допомагають спростити процес налаштування застосунку, а також стандартизують архітектуру та стиль написання коду, що важливо для можливого розширення функціоналу іншими розробниками.

Базою даних була обрана MongoDB в першу чергу через швидкодію, адже при накопиченні даних це відіграє важливу роль у продуктивності роботи. Більш того MongoDB являється достатньо новою технологією для розробки архітектурного рівня бази даних. А розгалуження бази даних на додаткову кількість вузлів, надає можливість ефективно зберігати та обробляти дані.

3.1 Мова програмування Java

Архітектура рівня бізнес-логіки (процесів системи) написана мовою високого рівня Java — об’єктно-орієнтованою мовою програмування.

Вибір платформи та реалізації продукту на мові Java зумовлений наявністю великої кількості технологій та фреймворків, створені для вирішень широкого спектра прикладних задач.

Однією з головних переваг Java є віртуальна машина (JVM), яка забезпечує кросплатформенність. Тобто, програму, створивши на Java та компілювавши її в байт-код, можливо запустити на будь-якій іншій платформі, яка буде підтримувати віртуальну машину Java. В такому випадку JVM буде рівнем абстракції між кодом та апаратним забезпеченням.

Ефективність програми безпосередньо пов’язана з пам’яттю, а пам’ять обмежена, тому вибір мови програмування Java являється кращим вибором, адже присутній в ній збирач сміття (англ. Garbage Collector) може знаходити об’єкти, на які програма більше не посилається, і видаляти їх. Хоч збирач сміття запобігає збереженню пам’яті, в розробці даного програмного продукту було виявлено недолік роботи з даною перевагою, тому продукт потребував оптимізації, генеруючи та імпортуючи дані до бази даних.

Так як Java є об’єктно-орієнтована мова високого рівня з жорсткою типізацію, вона має високу надійність роботи. Саме тому було виключено множинне наслідування та введено поняття інтерфейсу, що являє собою «контракт для класів», що будуть імплементувати його.

Наступним, одним з високих чинників надійності, є система для обробки помилок та винятків, які діляться на види:

- виняткові ситуації, які розробник має обов’язково обробити, наприклад: у випадку, якщо користувач увів недопустиму кількість символів;
- ситуації, які можуть виникнути при операціях таких як: переповнення пам’яті, яка виникла через неуважність програміста, чи робота з елементами масивом за його межами.

Java пропонує різні зручні інструменти для програмістів, такі як Java API – це набір команд та методів таких як наприклад: підключення до бази даних, що було пріоритетним в розробці даної інформаційної системи.

Парадигма ООП наділила мову такими властивостями як масштабованість, що дає змогу неодноразово розширювати розроблену систему. Масштабувати систему можна таким чином, аби при додаванні в систему нових компонентів не виникало змін у вже існуючих. Також, перевагою даного підходу є можливість використовування написаного коду багаторазово, що значно допомагає зменшити кількість повторюваності коду та загалом кількості написаного коду.

Java є розповсюдженою мовою програмування, так як вона широко використовується для написання різних типів програмних продуктів.

3.2. Фреймворк Apache Maven

Apache Maven — це фреймворк для автоматизації збірки проектів на основі опису їх структури в POM (англ. Plain Object Model) файлах. Це XML-файл, який містить інформацію про проект, конфігурацію, залежності (додаткові фреймворки) у вигляді, як зображено на рисунку 3.2.

```
<dependency>
      <groupId>org.mongodb</groupId>
      <artifactId>mongo-java-driver</artifactId>
      <version>3.8.0</version>
</dependency>
```

Рисунок 3.2 — Приклад опису залежностей в pom-файлі

Maven за замовчуванням містить стандартну структуру каталогів, які являються дуже практичними при сумісному програмуванні. Головний каталог scr містить в собі два каталоги: вхідний main та тестовий test, як зображено на рисунку 3.3.

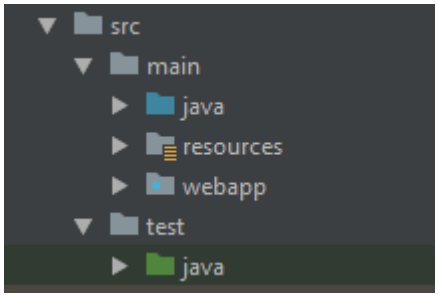

Рисунок 3.3 — Структура каталогів maven проектів

Зручним для використання при збірці проектів є набір фаз, які використовує Maven для керування життєвим циклом проекту. Найчастіше використовуваними в збірці даного проекту були використані такі фази, як clean — очищення проекту та install — встановлення програмного забезпечення в локальний репозиторій Maven, для доступу цього програмного забезпечення для інших проектів поточного користувача.

3.3. Мова розмітки HTML

Для розробки front-end частини даної системи та відображення її у вигляді Web-сторінки, було використано мову розмітки гіпертекстових документів HTML(англ. HyperText Markup Language). Дана технологія являється однією з розповсюджених та основних технологій для побудови будь-якої Web-сторінки.

Зручна структура написання сторінки на HTML дозволяє з легкістю додавати необхідні для масштабування системи сторінки, а наявність таких атрибутів, як id та class, являється зручним для рефакторингу.

3.4 Технологія JSP

Замість статичного вмісту, який є не змінюваним, був введений Java Servlet для створення динамічного веб-вмісту, який підпорядковується відповідно до запитів користувача, наприклад: відповідь на пошукові запити. Тим не менш, застосування Servlet для створення презентабельної HTML-сторінки не являється покращенням та

презентабельним, а зміна створеної HTML-сторінки не є гарним тоном в написанні презентаційного рівня програмного продукту.

Тому поява технології JSP (англ. JavaServlet Page) полегшує з'єднання динамічного та статичного веб-вмісту. JSP, подібно до популярного ASP (англ. Microsoft Active Server Pages), також забезпечує елегантний спосіб з'єднання динамічного та статичного веб-вмісту. Головна сторінка написана мовою розмітки HTML, а наявність спеціальних тегів забезпечує вставку фрагментів програмного кода на Java. Бізнес логіка та презентаційний рівні розумно розподілені.

JSP не являється заміною JavaServlet, це різні підходи до презентаційного рівня програмного продукту. В той момент, коли Servlet — це використання мови розмітки HTML в Java коді, то JSP — це використовування Java в середині HTML структури, тобто JSP є більш зручнішим способом для роботи з презентацією web-сторінки, аніж сервлет, але менш потужніший.

3.5. Мова розмітки CSS

CSS (англ. Cascading Style Sheets) — спеціальна мова, яка використовується для дизайну сторінок, написаних мовами розмітки даних.

Використання CSS полягає у забезпеченні презентабельного вигляду web-сторінкам. CSS працює з шрифтами, кольорами, висотою, шириною, позиціонуванням елементів, тобто надає структурованості вигляду контенту на web-сторінці.

3.6 Фреймворк JUnit

Для unit-тестування в даній інформаційній системі про бронювання (залізничних) квитків був використаний JUnit - це фреймворк, який використовує анотації для позначення методів як методів тестування та їх налаштування. У наведеній нижче Таблиці 1. наведено огляд найважливіших анотацій у JUnit для

версій 4.x та 5.x.

Таблиця 1. Анотації , які можна використовувати при тестуванні.

Junit 4	Опис
`@Test`	Визначає метод як метод тестування.
`@Before`	Виконується перед кожним тестом. Він використовується для підготовки тестової середовища (наприклад, читання вхідних даних, ініціалізація класу).
`@After`	Виконується після кожного тесту. Він використовується для очищення тестового середовища (наприклад, видалення тимчасових даних, відновлення за замовчуванням). Він також може заощадити пам'ять, очистивши структури пам'яті.
`@BeforeClass`	Виконується один раз, перед початком всіх тестів. Він використовується для виконання інтенсивних часу, наприклад, для підключення до бази даних. Методи, позначені цією анотацією, повинні бути визначені як статичні для роботи з JUnit.
`@AfterClass`	Виконується один раз, після завершення всіх тестів. Він використовується для виконання операцій очищення, наприклад, для відключення від бази даних. Для роботи з JUnit методи, позначені цим анотацією, повинні бути визначені як статичні.
`@Ignore` or `@Ignore("Why disabled")`	Позначає, що тест повинен бути відключений. Це корисно, коли базовий код був змінений і тестовий випадок ще не був адаптований. Або, якщо час виконання цього тесту занадто довгий для включення. Найкращою практикою є надання додаткового опису, чому тест відключений.
`@Test (expected = Exception.class)`	Не вдається, якщо метод не викине іменований виняток.
`@Test(timeout=100)`	Провалиться, якщо метод займає більше 100 мс.

Так як робота даної системи полягала в роботі з базами даних, тому для тестування необхідно було використовувати PowerMock, як генератор mock об'єктів під час виконання тестування.

3.7. Розподілена нереляційна база даних

В даній інформаційній системі про бронювання (залізничних) квитків використання NoSql пов’язана зі зберіганням великих об’ємів даних. Ці системи дуже масштабовані, можуть ефективно обробляти великі обсяги даних і підтримувати гнучкі, напівструктуровані дані. Системи NoSQL в основному базуються на простих

операціях читання-запису, де продуктивність є вирішальною проблемою. В даний час існує більше ста систем NoSQL з різними характеристиками моделі даних, різними API доступу до даних.

Бази даних NoSQL класифікуються на чотири основні категорії, кожна з яких відповідає різним видам завдань:

- ключ-значення (SimpleDB);
- документо-орієнтована база даних (CouchDB, MongoDB);
- широкий стовпчик або колонки-сім'я (Cassandra, HBase, Big Table);
- графи (InfoGrid, InfiniteGraph, Sones GraphDB).

Ця велика неоднорідність відкрила нові проблеми.

У світі дизайн даних NoSQL вимагає значних рішень моделювання, які впливають на значні вимоги до якості. Більш того, повної і повністю формалізованої процедури моделювання даних для систем NoSQL не існує. Розподіл даних відіграє значну роль у системах NoSQL , і багато дослідницьких підходів запровадили ефективні методи та алгоритми поширення даних [7].

3.7.1 Документно-орієнтована база даних MongoDB

У цій роботі ми визначаємо стратегію моделювання, яка підтримує використання документно-орієнтованої бази даних MongoDB.

Бази даних документів були, як випливає з їхньої назви, призначені для управління та зберігання документів. Ці документи кодуються у стандартному форматі обміну даними, наприклад, XML, JSON (нотація Javascript Option) або BSON (Binary JSON). На рисунку 3.4 продемонстровано в якому вигляді зберігаються дані в реляційній моделі даних та в моделі документів.

Рисунок 3.4 — Моделі реляційної та документно-орієнтованої моделі даних

Стовпець значень у базах даних документів містить напівструктуровані пари атрибутів ім'я / значення специфічних даних. Один стовпець може містити сотні таких атрибутів, а кількість і тип записаних атрибутів можуть змінюватися від рядка до рядка. Крім того, на відміну від простих сховищ ключ-значення, обидва ключі та значення повністю доступні для пошуку в базах даних документів.

Первинне використання: Бази даних документів є корисними для зберігання і керування великими колекціями великих даних, таких як текстові документи, повідомлення електронної пошти та XML-документи, а також концептуальні «документи», такі як денормалізовані (сукупні) подання об'єкта бази даних, наприклад продукт або клієнт. Вони також хороші для зберігання "загальних" даних загалом, тобто нерегулярних (напівструктурованих) даних, що вимагають широкого використання "нулів" в РСУБД (нулі є заповнювачами для відсутніх або неіснуючих значень).

3.7.2 Масштабування бази даних

Системи баз даних з великими наборами даних або додатками з високою пропускною здатністю можуть кинути виклик ємності одного сервера. Наприклад,

висока частота запитів може привести до вичерпання ресурсів процесора сервера. Розміри робочого набору, що перевищують обсяг оперативної пам'яті системи, підкреслюють ємність вводу-виводу дисків.

Виділяють два метода масштабування:

- горизонтальне масштабування;
- вертикальне масштабування.

Під вертикальним масштабуванням розуміють нарощування потужності устаткування (фізичного чи віртуального) чи програмного забезпечення за допомогою збільшення кількості ресурсів. Обмеження в доступній технології можуть обмежити потужність однієї машини для даної робочої навантаження. Крім того, хмарні провайдери мають жорсткі обмеження на основі доступних конфігурацій обладнання. В результаті існує практичний максимум для вертикального масштабування.

В даній роботі використовувалось горизонтальне масштабування, яке включає в себе поділ набору системних даних і навантаження на кілька серверів, додавання додаткових серверів для збільшення ємності в міру необхідності. Хоча загальна швидкість або ємність одного комп'ютера може бути не високою, кожна машина обробляє підмножина загальної робочої навантаження, потенційно забезпечуючи кращу ефективність, ніж один високошвидкісний сервер з високою пропускною здатністю. Розширення ємності розгортання вимагає тільки додавання додаткових серверів в міру необхідності, що може бути дешевше в порівнянні з високопродуктивним обладнанням для одного комп'ютера. Компромісом є підвищена складність інфраструктури та обслуговування для розгортання.

MongoDB підтримує горизонтальне масштабування за допомогою шардінга.

3.7.3. Розподілення бази даних MongoDB на кластери

MongoDB підтримує горизонтальне масштабування за допомогою шардінга.

Термін "sharding" в цілому застосовується до баз даних, ідея полягає в тому, що єдиного комп'ютера ніколи не може бути достатньо для зберігання всіх даних.

Sharding відбувається при розподілі бази даних на окремі шматки, які знаходяться на різних машинах.

Простим прикладом може бути: припустимо, що у компанії є машини, які можуть зберігати до 2 мільйонів елементів даних клієнтів. Тепер бізнес досягає цієї точки розриву і, швидше за все, скоро перевищить 2,5 мільйона користувачів. Отже, вони вирішили розбити свою базу даних на дві, як зображено на рисунку 3.5.

Рисунок 3.5 — Демонстрація прикладу sharding процесу (фрагментування однієї бази даних у дві)

Для архітекторів програмного забезпечення хвилювання про MongoDB було не стільки в його гнучкій схемі, скільки в її вбудованій підтримці.

За допомогою декількох простих правил і машин, підключених, можна бути готовими запустити кластер MongoDB в найкоротші терміни.

На рисунку 3.6 показано, як це виглядає у типовому розгортанні веб-застосунку:

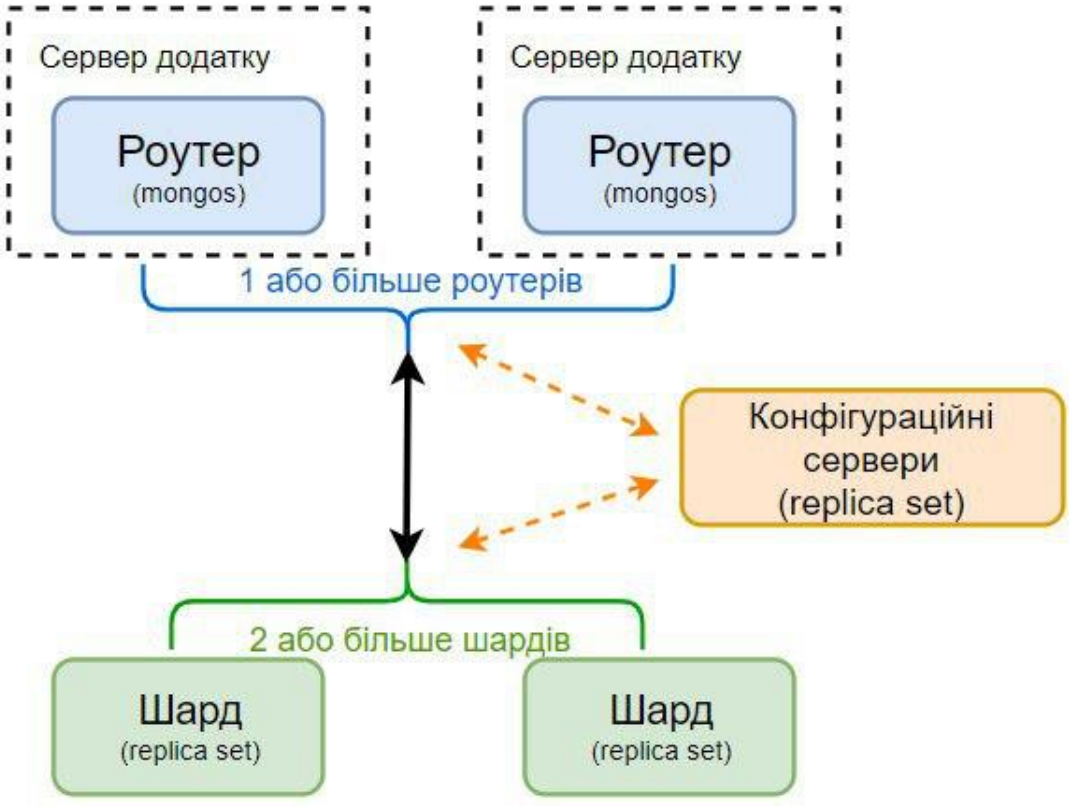

Рисунок 3.6 — Взаємодія компонентів в ізольованому кластері

Найкраща частина процесу фрагментації MongoDB полягає в тому, що навіть балансування фрагментів відбувається автоматично. Тобто, якщо у вас є п'ять фрагментів, і два з них майже порожні, можна заавтоматизувати MongoDB таким чином, аби всі фрагменти були однаково повні.

Як розробнику та адміністратору, не потрібно багато турбуватися, оскільки MongoDB виконує більшу частину важкої роботи. Те ж саме стосується і часткової відмови вузлів: якщо у кластері є правильно налаштовані та запущені набори реплік, часткові відключення не вплинуть на час роботи системи.

При розгортанні sharding процесу потрібно вибрати ключ з колекції та розділити дані за допомогою значення ключа. Цей ключ називається ключем shard, який визначає, як розподіляти документи з колекції між різними фрагментами кластера.

Ключ shard — це поле, яке існує в кожному документі в колекції і може бути

індексованим або індексованим складовим полем. MongoDB виконує розділи даних в колекції, використовуючи різні діапазони або блоки значень ключів shard.

Кожен діапазон або блок визначає чи не дублюється значення ключів shard. MongoDB поширює блоки та їх документи серед shard в кластер. MongoDB також розповсюджує документи відповідно до діапазону значень у ключі shard.

Щоб підвищити та оптимізувати продуктивність, функціонування та можливості бази даних, потрібно вибрати правильний ключ Shard.

Вибір відповідного ключа клавіатури залежить від двох факторів:

- схема даних;
- спосіб застосування до запиту бази даних і виконання операцій запису.

Висновки до розділу:

При розробці програмного продукту важливим чинником є правильний вибір засобів програмної реалізації. Вибір стеку технологій може сильно впливати на продуктивність системи, а особливо на такі фактори, як швидкість обробки запитів, надійність системи та стабільність системи при навантаженнях.

У розділі обґрунтовано технології для розробки архітектурного рівня бази даних та описано головні принципи їх використання. Винесено, що при використанні нереляційної бази даних, система потребує горизонтального масштабування, яке забезпечить стабільності системи при великих навантаженнях на систему.

Також зроблено огляд головних технологій, які використовувались при реалізації програмного продукту, зокрема це мова програмування Java, фреймворк збірки проектів Apache Maven, та фреймворки для тестування, головні технології розробки front-end частини.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

Для автоматизації інформаційної системи про бронювання (залізничних) квитків необхідно було розробити масштабовану систему з використанням нереляційної бази даних, як ефективного рішення для роботи та обробки великих даних, і яка буде надавати доступ до різних модулів системи відповідно до ролі та наданих прав користувачу.

При розробці даної системи було обрано вибрано клієнт-серверну архітектуру, так як запити від одного і більше клієнтів будуть оброблятися одним централізованим сервером, і працювати зі сховищем інформації як з єдиним централізованим сховищем, так як сервер працює в прозорому режимі, не знаючи про існування декількох кластерів сховища даних.

Доступ до різних рівнів функціоналу та даних для користувача залежить від його ролі в системі.

Архітектурне рішення рівня бази даних зумовлено обробкою великих даних. Крім того, дане рішення являється достатньо нерозповсюдженим та новим рішенням у розробці інформаційних систем, тому було розроблено використання двох рівнів баз даних, для даної системи задля обґрунтування поставленої задачі. Окрім цього, поставлена задача, розробки нереляційної розподіленої бази даних, забезпечує гнучкий підхід та відмово-стійкість, таким чином забезпечивши систему надійністю зберігання даних.

4.1. Структура програмного забезпечення

Під час реалізації архітектури рівня бізнес-логіки програмного продукту використовувались різні шаблони GoF такі як: Фабричний метод, Команда, Одиночка, Будівник.

Структура папок головних каталогів має вигляд згідно до Maven фреймворку, як показано на рисунку 4.1.

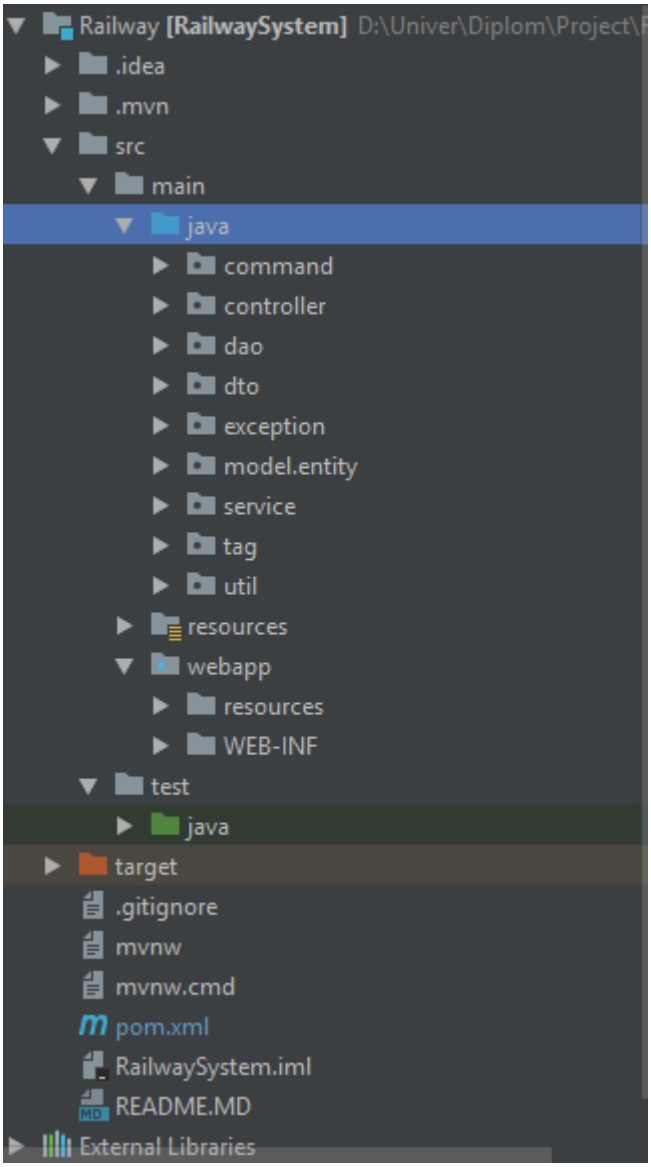

Рисунок 4.1 — Структура папок в програмному продукті

Головними каталогами є: src, target, .idea та файлів .gitignore, External Libraries, Railway.iml та pom.xml. Папка scr містить в собі логіку проекту, та поділяється на дві головні папки main, яка в свою чергу містить в собі java, resources, webapp, та test, в якій знаходяться покриття всіх методів unit-тестами.

Папка target зберігає в собі скомпільовані файли проекту, які в подальшому розгортаються на сервер, .idea містить інформацію про проект , та являється згенерованим програмою модуль середовища розробки, External Libraries — це модуль, в якому зберігається набір підключених бібліотек, які необхідні були для

реалізації даного проекту, файл Railway.iml створюється автоматично при імпорті проекту в середовище розробки, також дуже зручно використовувати при наступному імпорті даного проекту в IntelliJ IDEA повторно, чи на іншому сервері, . gitignore відслідковує інтегровані файли та pom.xml — це єдиний файл конфігурації проекту в Maven фреймворку, який містить головну необхідну інформацію щодо збірки проекту та залежності.

Папка main поділяється на:

- java, яка містить в собі логіку проекту архітектуру рівня бізнес-логіки, та конфігурацію підключення та роботи з базою даних, а також роботу з презентаційним рівнем, тобто з інтерфейсом та запитами користувача;
- resources містить конфігураційні файли для роботи з базами даних, та для взаємодії з інтерфейсом (наприклад, зберігаються різні типи повідомлень, як зображено на рисунку 4.2) ;
- webapp містить реалізацію презентаційного рівня проекту.

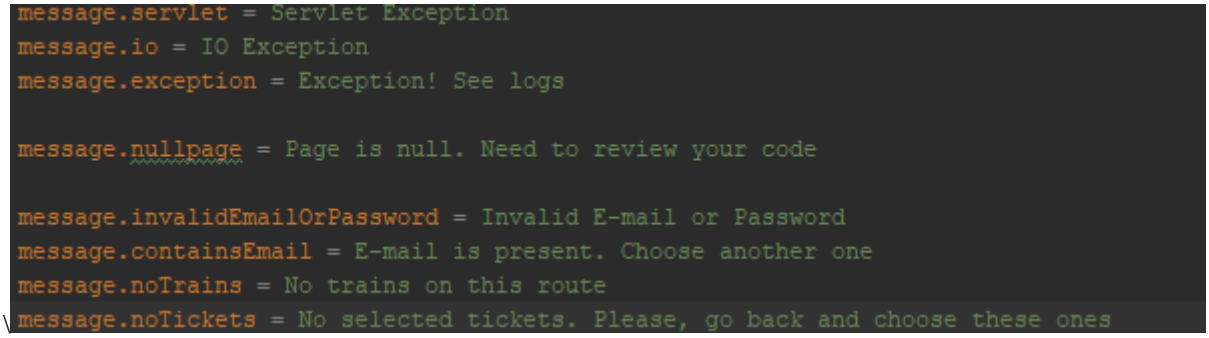

Рисунок 4.2 — Повідомлення про помилку

Папка java містить в собі такі модулі як на рисунку 4.3 :

- Controller, який відповідає за приймання та відповідь запитів(get/post), який відбувається за протоколом HTTP;
- Command, в якому відбувається обробка запитів;
- Service — це модуль бізнес-логіки;
- util, dao, dto — модулі, які відповідають за взаємодію з базою даних.

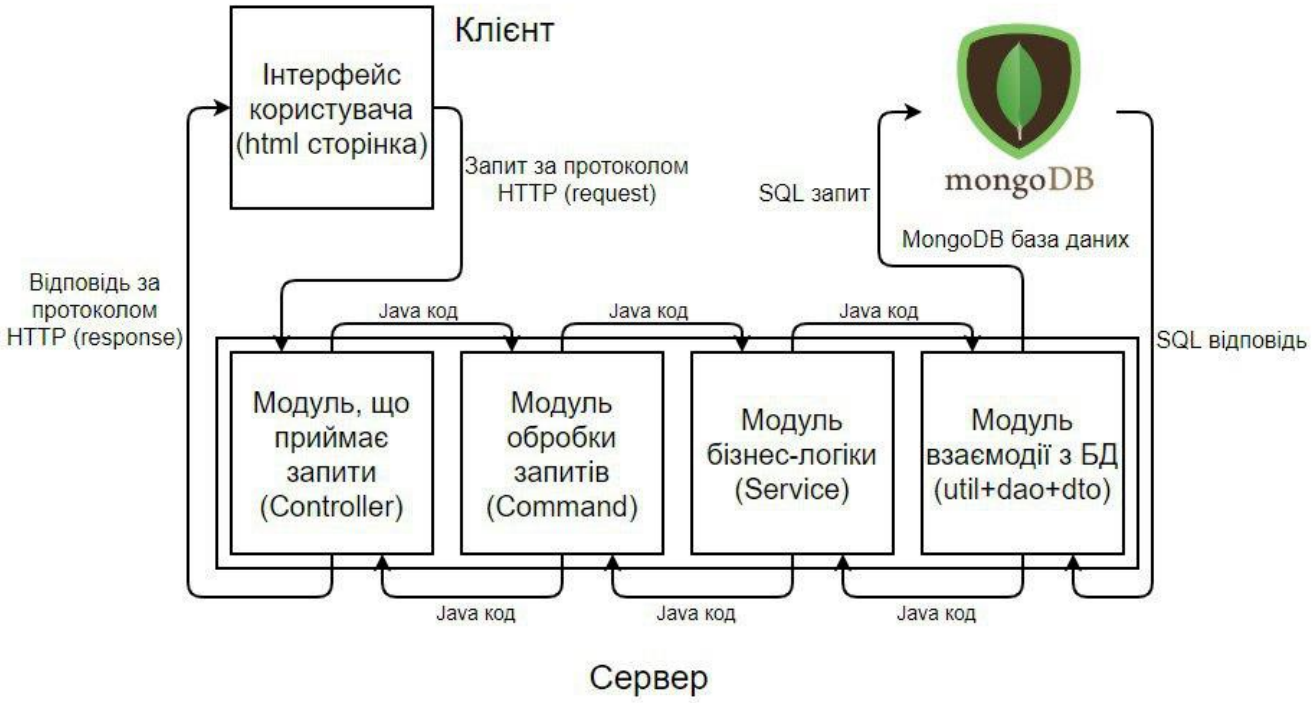

Рисунок 4.3 — Схема взаємодії модулів

4.2. Опис структури бази даних

Під час постановки задачі було вирішено розробити архітектурний рівень бази даних, який демонструє новий підхід для роботи з великими даними. Було обрано нереляційну розподілену базу даних MongoDB, для покращення роботи з інформаційною системою, а саме для зручного масштабування системи, при необхідності.

MongoDB для горизонтального масштабування використовую шардінг, сенс якого полягає в розподілу бази даних на окремі частини так, аби кожну його частину можна було винести на окремі сервера. Однак реплікація даних є обов’язковим при шардінг процесу в MongoDB. Репліка сет складається з сервера конфігурації та шарду. Кожний сервер являє собою запущений процес mongod.

Для розробки розподіленої бази даних в MongoDB створено два шарда та один конфігураційний сервер. Кожен шард та конфігураційний сервер становить з себе три вузла, для відмово-стійкості та зручного автоматичного відновлення даних, при виникненні проблем з одним з вузлів.

Команди для створення :

- configsvr позначає, що даний сервер використовується як конфігураційний;
- replSet позначає, до якого саме набору належить сервер;
- shardsvr позначає, що сервер буде використовуватись як шард.

На рисунку 4.4 продемонстровано приклад команд, якими було створено три конфігураційна сервері, а на рисунку 4.5 ініціалізації трьох серверів, як єдиного.

```
mongod --configsvr --replSet "myrepl" --dbpath "C:\Program Files\MongoDB\Server\4.0\data\db_config_1" --port 27019
mongod --configsvr --replSet "myrepl" --dbpath "C:\Program Files\MongoDB\Server\4.0\data\db_config_2" --port 27020
mongod --configsvr --replSet "myrepl" --dbpath "C:\Program Files\MongoDB\Server\4.0\data\db_config_3" --port 27021
```

Рисунок 4.4 — Створення 3-х конфігураційних серверів

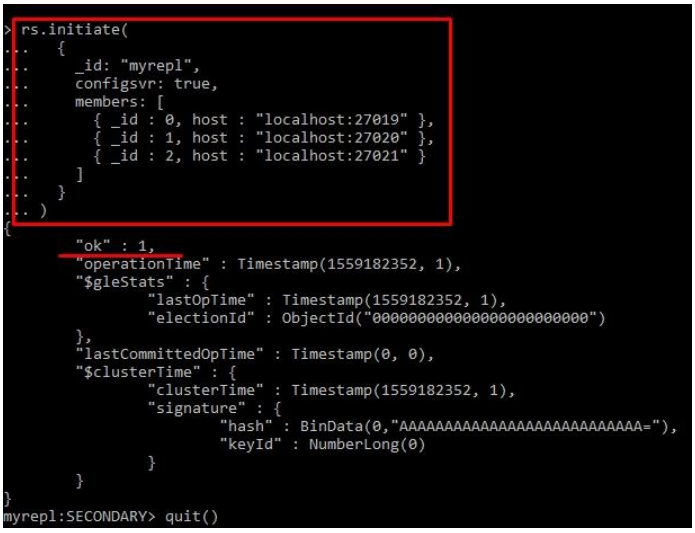

Рисунок 4.5 — Ініціалізація трьох серверів конфігурації

Такий самий процес був створений і для двох шардів, які вміщають в себе по три вузли, два з яких є вторинними і один первинний, з якого відбувається реплікація даних.

Робота з вузлами бази даних відбувається через центральний сервер (роутер). Налаштування розробленої архітектурної моделі полягало в налаштуванні моделі таким чином, аби сервіс працював з базою даних, як з одним централізованим ресурсом.

На рисунку 4.6 показано головні команди для налаштування роутера та додавання всіх вузлів(шардів) в єдиний головний сервер, через який і буде

відбуватися обробка запитів до даних. Саме на рівні роутера доступу до даних відбувається розподіл даних на різні репліка-сети, використовуючи для цього sharding key.

```
mongos --configdb "myrepl"/localhost:27019,localhost:27020,localhost:27021 -port 47019
mongo --host localhost --port 47019

sh.addShard( "shardrepl/localhost:37019")
sh.addShard( "shardrepl/localhost:37020")
sh.addShard( "shardrepl/localhost:37021")
sh.addShard( "shardrepl2/localhost:37022")
sh.addShard( "shardrepl2/localhost:37023")
sh.addShard( "shardrepl2/localhost:37024")
sh.enableSharding("railway_system")
sh.shardCollection("railway_system.users", { "id" : "hashed" } )
sh.shardCollection("railway_system.requests_sequence", { "id" : "hashed" } )
sh.shardCollection("railway_system.users_sequence", { "id" : "hashed" } )
sh.shardCollection("railway_system.routes", { "id" : "hashed" } )
sh.shardCollection("railway_system.requests", { "id" : "hashed" } )
sh.shardCollection("railway_system.stations", { "id" : "hashed" } )
sh.shardCollection("railway_system.prices", { "id" : "hashed" } )
sh.shardCollection("railway_system.trains", { "id" : "hashed" } )
db.users_sequence.insert({id: "users_sequence", sequence: 2})
db.requests_sequence.insert({id: "requests_sequence", sequence: 1})
```

Рисунок 4.6 — Команди для додавання шардів до головного роутера

Реплікація даних по вузлах відбувається на рівні репліка-сету, тобто після розподілу інформації між двома(в даному випадку) репліка-сетами, наступним обов’язковим кроком є реплікація первинного вузла та двох вторинних, як зображено на рисунку

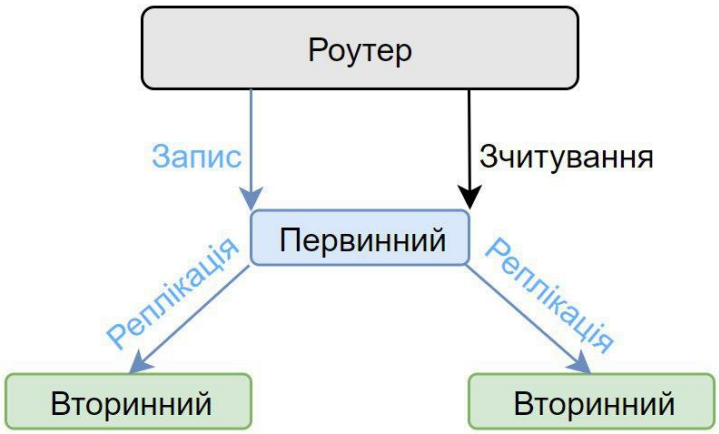

Рисунок 4.7 — Реплікація даних в кластері

Головною перевагою даної архітектурної моделі рівня бази даних полягає в горизонтальній масштабованості бази даних, тому з легкістю можна додати додатковий репліка-сет, не пошкодивши вже існуючі дані на різних кластерах.

Для масштабування системи необхідно:

- створити новий кластер з потрібною кількістю вузлів в ній;
- додати даний кластер до роутера.

Дана інформаційна система зручна для масштабування, як рівня бази даних, так і рівня бізнес-логіки. Розроблена система взаємодіє не лише з поставленою задачею роботи з нереляційною розподіленої базу даних, а взаємодіє з реляційною базою даних MySQL для наглядного порівняння. На рисунку 4.8 показана схема бази даних, розроблена мовою SQL(англ. Structured query language).

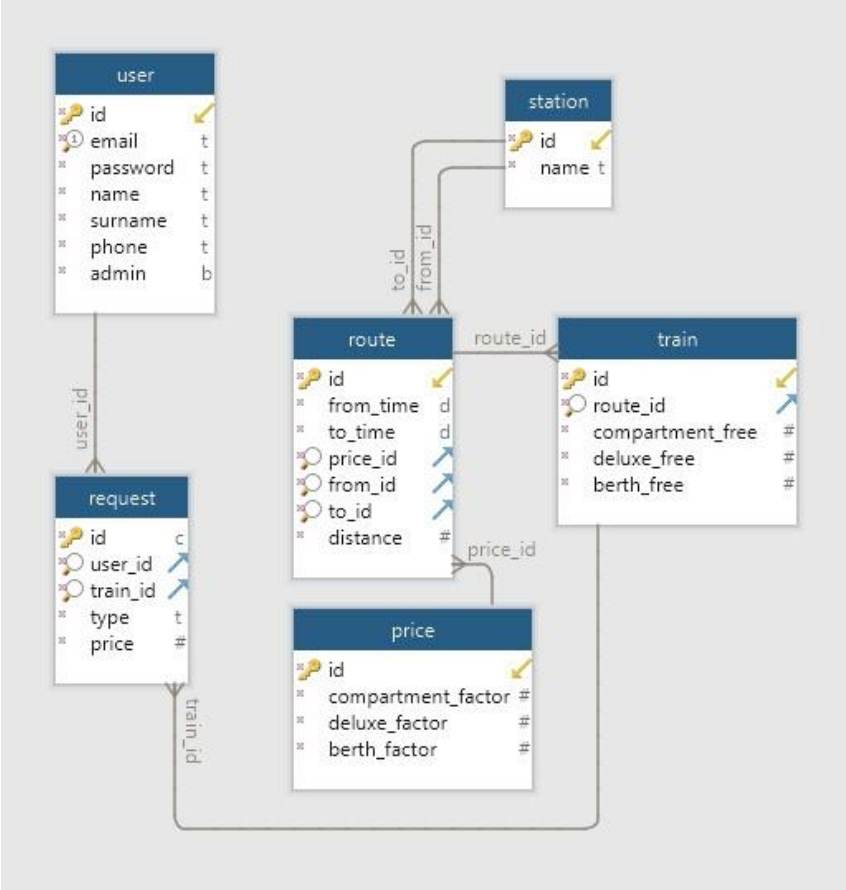

Рисунок 4.8 — Схема структури бази даних MySQL

Зручний підхід, який забезпечує використання двох різних архітектурних підходів для роботи з рівнем бази даних налаштовується при запуску додатку лише

за допомогою однієї конфігурації: config.profile.database=mysql/mongodb.

Головною перевагою використання бази даних MongoDB в даному програмному проекті є взаємодія з даними через програмний код, а не через з використанням SQL-запитів, яких не можливо уникнути при роботі з розподіленою базою даних. Також не менш важливим є те, що при необхідній зміні структури бази даних потрібно відобразити зміни лише в програмному коді.

4.3 Опис роботи з даними системи

Для роботи з даними було використано шаблони дизайну роботи з даними такі як : DAO(англ. Data Transfer Object) та DTO(англ. Data Access Object).

DTO використовується для передачі даних між класами та модулями програми, і являє собою звичайний об’єкт, який зберігає дані. В даному програмному продукті dto використано для зберігання об’єктів таких, як наприклад: Ticket, який містить інформацію про квиток та TrainRoute, який містить інформацію про залізничний маршрут. Структура класів зображена на рисунку 4.9.

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TrainRoute {
    private Long trainId;
    private Long routeId;

    private Integer compartmentF
    private Integer deluxeFree;
    private Integer berthFree;

    private String fromDate;
    private String toDate;

    private String fromCity;
    private String toCity;

    private Double compartmentPr
    private Double deluxePrice;
    private Double berthPrice;

    private Double distance;
}
```

```
import ...

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Ticket {
    private Long requestId;
    private Long trainId;
    private Long userId;

    private String name;
    private String surname;

    private String fromDate;
    private String toDate;
    private String fromCity;
    private String toCity;

    private String typePlace;
    private Integer max;

    private Double price;
}
```

Рисунок 4.9 — Структура dto на прикладі квитків та залізничних маршрутів

DAO інкапсулює логіку для отримання, збереження та оновлення даних у

сховищі даних. Тобто має операції CRUD, такі як створення, збереження, зчитування, видалення та редагування. В даному програмному продукті на прикладі Tickets, dao інкапсулює методи, які продемонстровано на рисунку 4.8.

```
public interface TrainDAO {
    /**
     * Find all TRAINS in DB
     */
    List<Train> findAll();

    /**
     * Find TRAINS by given ID of ROU
     */
    List<Train> findByRoute(Long route

    /**
     * Find TRAIN by ID
     */
    Train findById(Long id);

    /**
     * Insert new TRAIN
     */
    Train create(Train train);

    /**
     * Update TRAIN
     */
    Train update(Train train);

    /**
     * Delete TRAIN
     */
    void delete(Train train);

}
```

Рисунок 4.10 — Структура dao в проекті на прикладі доступу до квитків

Перевага шару DAO полягає в тому, що при необхідності зміни механізму персистентності, необхідно лише змінити шар DAO, а не всі домени в логіці домену, де використовується шар DAO.

Перевагою DTO-шару є те, що він додає значну гнучкість шару обслуговування, а потім і до дизайну всього додатка. Наприклад, зміна вимог, що змушує перейти до іншої кількості даних, не впливає на рівень обслуговування або навіть на домен. Необхідно перенести зміни до класу DTO, додаючи нову властивість, але залишивши загальний інтерфейс шару служби незмінним.

Головна задача даної інформаційної системи полягала в роботі з великими

даними. Для цього необхідно було спочатку згенерувати великий об’єм даних, для системи.

На рисунку 4.10 продемонстровано запити для імпорту випадково генерованих даних в json-файл (англ. JavaScript Object Notation) на головний сервер (роутер).

mongodb_import_railway_system.sh

```
#!/usr/bin/env sh

mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_0.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_1.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_2.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_3.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_4.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_5.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_6.json" --
mongoimport --host localhost --port 47019 --db railway_system --collection users --type json --file "../json/users_7.json" --
```

Рисунок 4.10 — Імпорт json-файлів з даними до бази даних

В даній базі даних було за імпортовано понад 130 тис користувачів системи. На рисунку 4.11 продемонстровано вивід кількості користувачів в даній базі даних через термінал cmd та використання команди db.users.count().

На рисунку 4.12 продемонстровано вивід кількості користувачів в системі саме на веб-застосунку, для зручного обігу кількістю користувачів в системі.

Command Prompt - mongo --port 47019 -u "root" -p "root" --authenticationDatabase "admin"

```
l.com", "password" : "63a9f0ea7bb98050796b649e85481845", "admin" : false }
mongos> db.users.find({id:130086})
{ "_id" : ObjectId("5cef3e93b85446000daa5a24"), "id" : 130086, "name" : "Jermaine", "surname" : "Hyatt", "phone" : "(012) 345-67-89"
n", "password" : "63a9f0ea7bb98050796b649e85481845", "admin" : false }
mongos> db.users.find({id:130087})
{ "_id" : ObjectId("5cef3e93b85446000daa5a20"), "id" : 130087, "name" : "Merle", "surname" : "O'Hara", "phone" : "(012) 345-67-89",
"password" : "63a9f0ea7bb98050796b649e85481845", "admin" : false }
mongos> db.users.find({id:0})
mongos> db.users.find({id:1})
{ "_id" : ObjectId("5cef3e5ab85446000da85dac"), "id" : 1, "name" : "Anissa", "surname" : "Hansen", "phone" : "(012) 345-67-89", "ema
ssword" : "63a9f0ea7bb98050796b649e85481845", "admin" : false }
mongos> db.users.count();
130161
mongos> db.users.find({id:130088})
{ "_id" : ObjectId("5cef3e93b85446000daa5a22"), "id" : 130088, "name" : "Shanika", "surname" : "Torphy", "phone" : "(012) 345-67-89"
n", "password" : "63a9f0ea7bb98050796b649e85481845", "admin" : false }
```

Рисунок 4.11 — Кількість користувачів в системі(термінал)

Для демонстрації, що дані в базі даних співпадають з даними, які виводяться на веб-сторінку додатку, показано на рисунках 4.13 та 4.12 користувачів системи, відповідно спочатку вивід в консолі за допомогою mongos.

Рисунок 4.12 — Кількість користувачів в системі(термінал)

Mongos для "MongoDB Shard" - це служба маршрутизації для конфігурацій Shard MongoDB, що обробляє запити з прикладного рівня, і визначає розташування цих даних у кластері sharded, щоб завершити ці операції. З точки зору програми примірник mongos поводиться однаково з будь-яким іншим екземпляром MongoDB[1].

Рисунок 4.13 — Вивід користувачів системи в консоль

Вивід відбувався по id окремо за допомогою команди:

— db.users.find({id: <id>}), так як функція db.users.find() бере значення з декількох вузлів одночасно і тому отриманні дані виводяться не послідовно.

Висновки до розділу:

В даному розділі описано структуру бізнес-логіки та бази даних програмного продукту. Розглянуто архітектуру системи та взаємодію модулів як між собою так і з системою в цілому.

Основна інформація даного розділу полягала в описі розробки архітектури рівня бази даних, тобто розробка нереляційної бази даних, розподілення системи завдяки горизонтальному масштабуванню. Проаналізовано роботу з даними системи, використовуючи інтерфейс та термінал.

Для роботи з базами даних та їх об’єктами описано шаблони, які допомагають розробити правильну структуру роботи, аби забезпечити можливу зміну в структурі даних, яке може призвести до неочікуваних проблем.

Досліджено роботу даної системи з різними видами баз даних: реляційною та нереляційною. Проаналізовано роботу з кожною та зроблено висновки з перевагами та недоліками кожної.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для роботи з системою є перелік головних вимог, які необхідні при роботі з даною системою. По-перше підключення до мережі Інтернет являється найважливішою вимогою.

При переході на сайт користувач одразу потрапляє на сторінку авторизації, як зображено на рисунку 5.1 , де одразу і відбувається доступ до різних модулів, при наявності потрібного рівня доступу до системи.

Також для кожного користувача системи одразу надається можливість вибору мови відображення сторінки, як зображено на рисунку 5.1. Для роботи з системою надається дві головні у використанні мови: українська та англійська

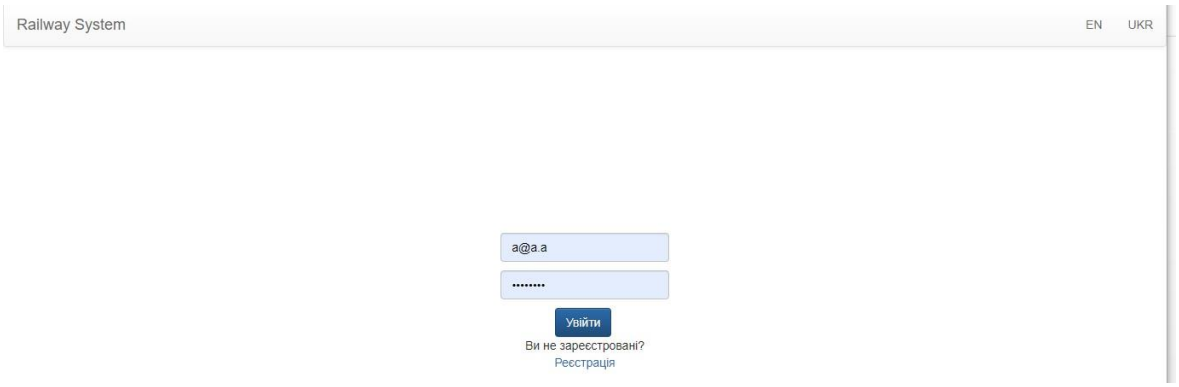

Рисунок 5.1 — Головна web-сторінка системи

5.1. Системні вимоги

Аби користувач міг працювати з даною інформаційною системою йому необхідні мінімальні характеристики апаратного забезпечення, які наведені в таблиці нижче 3.

Таблиця 3. Вимоги до апаратного забезпечення клієнта

Пристрій	Характеристика
Процесор	Intel ® Core ™ 2 / 2 Duo / Pentium ® / Celeron ® / Xeon™ / i3 / i5 / i7 чи AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™ з тактовою частотою не нижче 1.5 GHz.
Оперативна пам'ять (RAM − Random Access Memory)	Рекомендовано не менше 1 RAM
Швидкість з'єднання з інтернет	Рекомендовано не менше 128 кб/сек
Операційна система	Mac OS 9, Windows XP, Ubuntu 9

Підтримувані апаратні архітектури:

- 32-розрядна (x86);
- 64-розрядна (x64)

5.2. Ролі користувачів в системі

У розробленій інформаційній системі існує три ролі, які необхідні для повного користування даною системою:

- адміністратор;
- користувач;
- відповідальний за інформацію по квиткам.

Діаграма прецедентів, зображена на рисунку 5.2 демонструє головні функції кожної ролі в системі.

Згідно з діаграмою прецедентів, кожен користувач системи має свої, не сумісні з іншими ролями функції для роботи з даною системою.

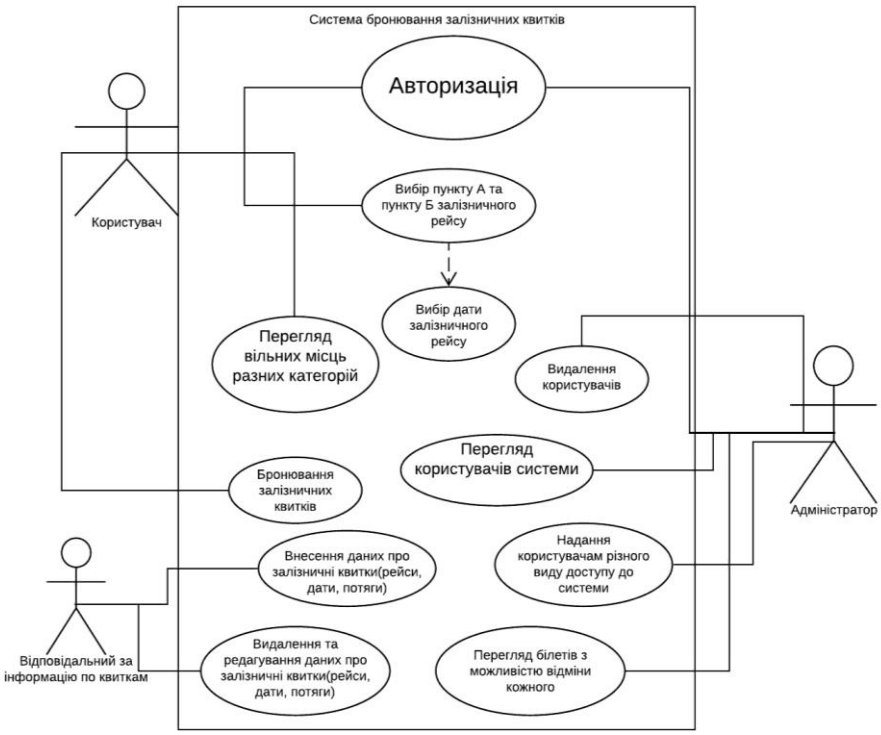

Рисунок 5.2 — Діаграма прецедентів

Таким чином, адміністратор має найважливіші функції для керування та обігом даних. Користувач системи має функції, які мають обмежений функціонал, через рівень доступу до модулів. Відповідальний за інформацію по квиткам має найважливіші функції, адже, саме від нього залежить, які саме дані будуть використані користувачами, які дані будуть внесені до бази даних.

5.3. Сценарії роботи користувачів розробленої системи

Авторизація в систему являється обов’язковою для бронювання квитків. Лише авторизований користувач, має доступ до системи, в такому випадку він потрапляє до наступної сторінки з різним функціоналом, в залежності від ролі даного користувача в системі.

При невдалій спробі авторизуватись, користувач бачить сповіщення, як зображено на рисунку 5.3, про не валідні дані, які він намагається ввести, так само відображується сповіщення про обов’язано необхідні заповнені поля для входу в

систему зображено на рисунку 5.4.

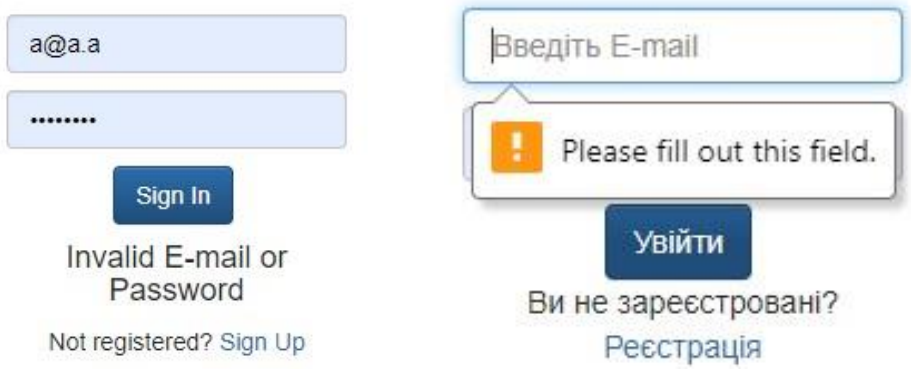

Рисунок 5.3, 5.4 — Сповіщення системи про неправильний ввід даних та обов’язково заповнені поля для входу

При відсутності користувача в системі, він може пройти процес реєстрації в систему, як зображено на рисунку 5.5. Для цього йому необхідно було на головній сторінці натиснути на посилання Реєстрація/Sing Up(при використанні англійської версії системи) , зображеному на рисунках 5.3 та 5.4

Рисунок 5.5 — Реєстрація користувача в систему

Після реєстрації чи авторизації користувача в системі, користувач потрапляє на web-сторінку та бачить у хедері сторінки привітання з використанням його імені, таким чином, користувач може впевнитись, що авторизувався саме на в свій кабінет.

Після проходження авторизації, користувач переходить на сторінку з вибором усі необхідних для пошуку даних, які необхідно ввести, для пошуку конкретного рейсу (рисунок 5.6).

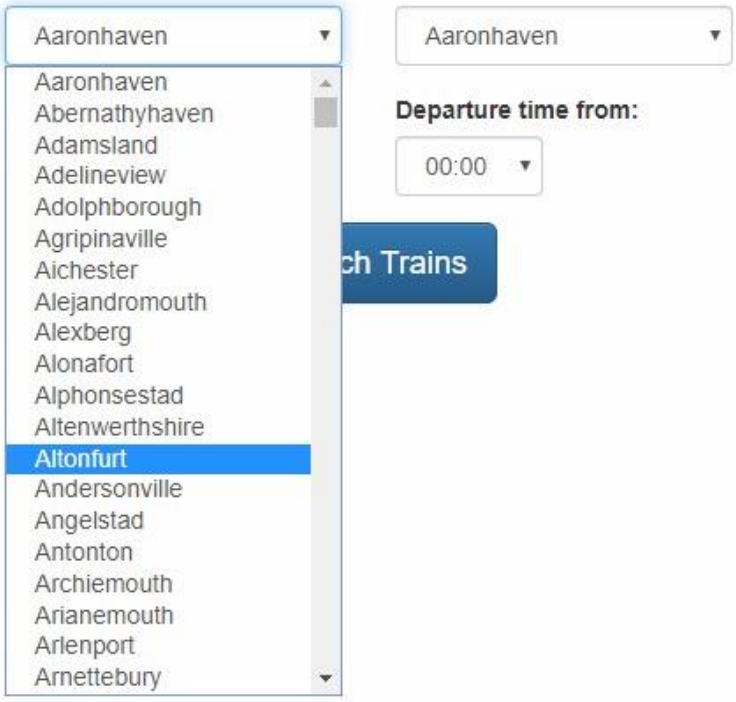

Рисунок 5.6 — Функціонал вибору конкретного залізничного квитка

Крім вибору міст, користувач обирає, дату необхідну для пошуку вільного для бронювання залізничного квитка.

По можливості, користувач вказує час, який йому зручний (рисунок 5.7). При відсутності вказаного користувачем часу, система показує всі вільні для бронювання квитки за конкретно обраний день.

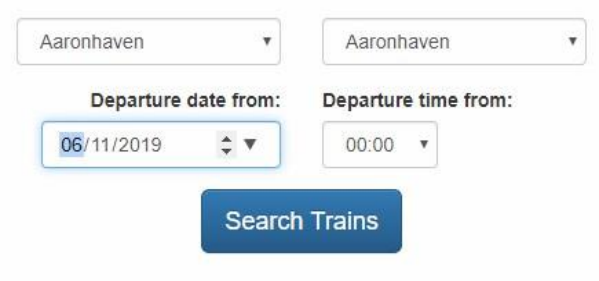

Рисунок 5.7 — Необхідні для пошуку залізничного квитка заповнені дані

Після знаходження необхідного залізничного квитка, користувач може обрати місця згідно класу пасажирського місця, так користувач може брати будь-яке вільне місце з класу: A, B, C (рисунок 5.8).

Ціна будь-якого квитка залежить від обраного користувачем класу пасажирського місця.

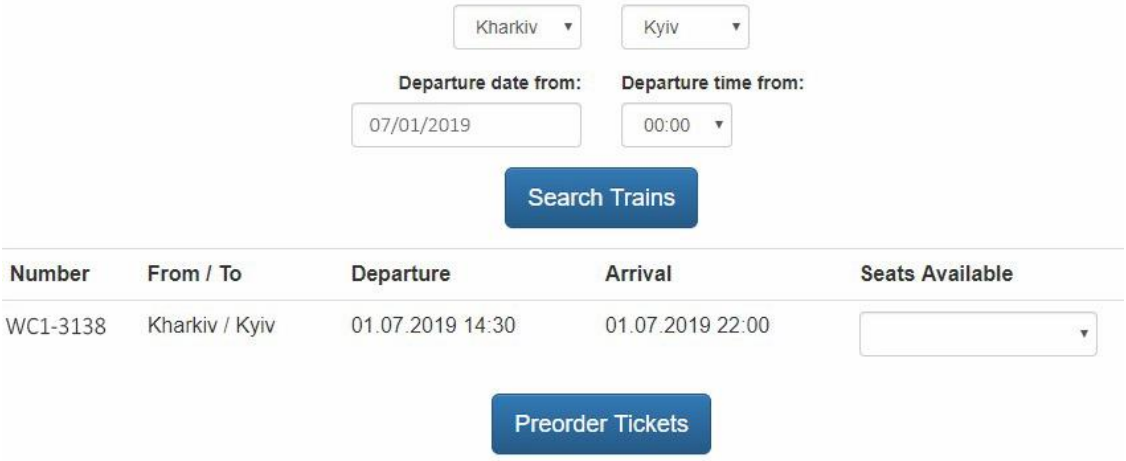

Рисунок 5.8 — Сторінка з вибором необхідного залізничного квитка

Після вибору місць, доступна функція кількості залізничних квитків які необхідні користувачу(рисунок 5.9).

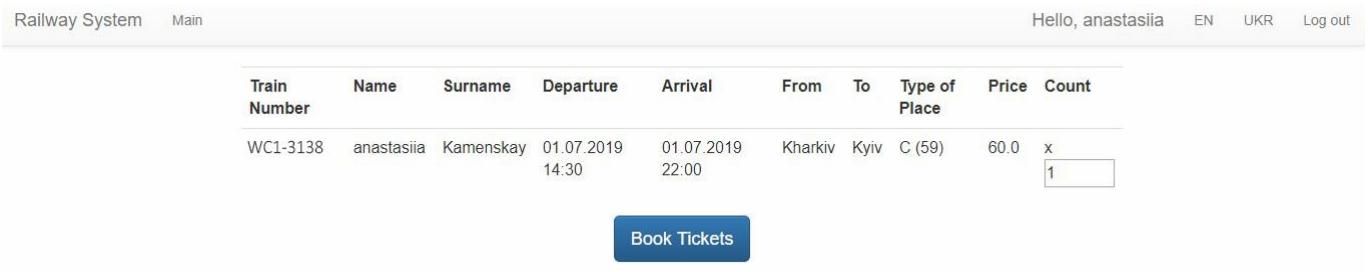

Рисунок 5.9 — Сторінка з вибором кількості місць для бронювання квитків

Адміністратор одразу після авторизації в системі потрапляє на сторінку, в якій відображаються користувачі системи (рисунок 5.10). Адміністратор має можливість перегляду усіх користувачів системи. Через велику кількість користувачів в системі, було розроблено пошук по користувачам, який дає можливість шукати в системі користувачів по:

- електронній адресі;
- імені та прізвищу користувача.

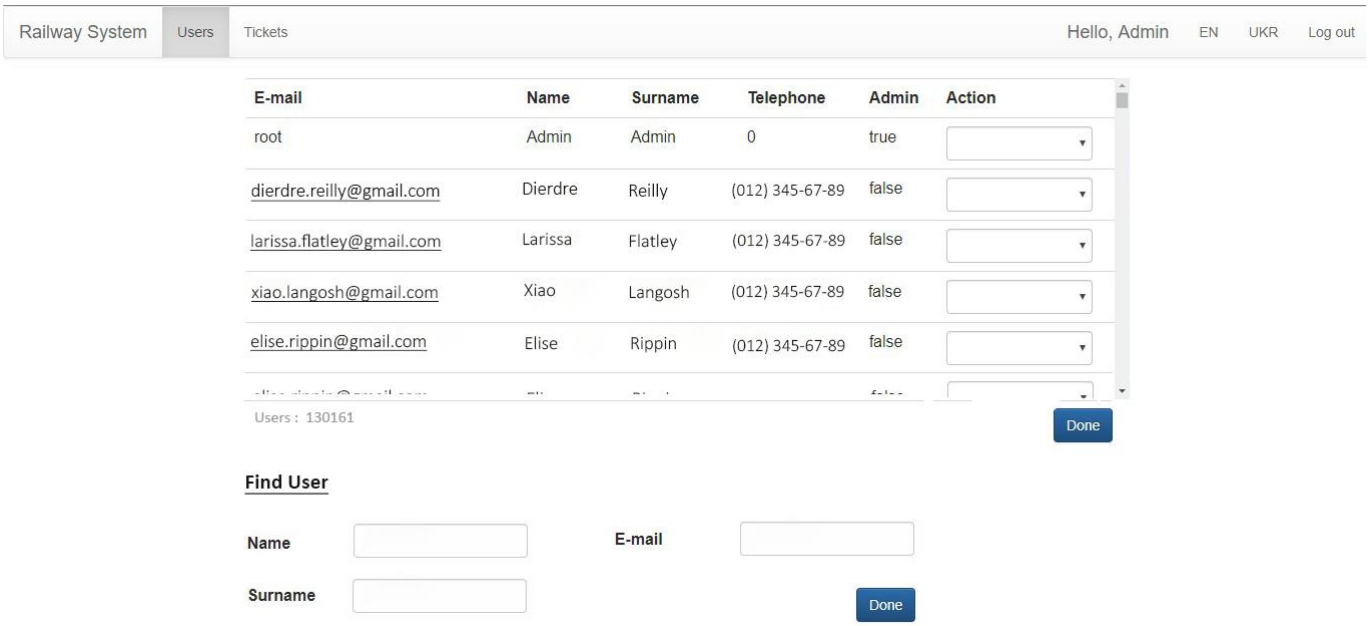

Рисунок 5.10 — Головна сторінка адміністратора

На рисунку 5.10 також видно, що перший в списку користувачів системи є адміністратор даної інформаційної системи.

Адміністратор також має можливість надавати різні рівні доступу до додатку кожному користувачу, як продемонстровано на рисунку 5.11. В списку на різні рівні доступу до системи є :

- зробити адміністратором;
- зробити користувачем;
- видалити користувача з системи.

Рисунок 5.11 — Надання користувачу нового рівня доступу в систему

Переходячи на сторінку з інформацією про заброньовані квитки, адміністратор системи бачить календар для вибору дати виїзду, по якій відбувається пошук заброньованих квитків. Після вибору конкретної дати на сторінці з'являється список з залізничними квитками. Одним з головних функцій адміністратора являється можливість видалення квитків з системи, як усіх квитків по заданому дню, так і конкретного (рисунок 5.12).

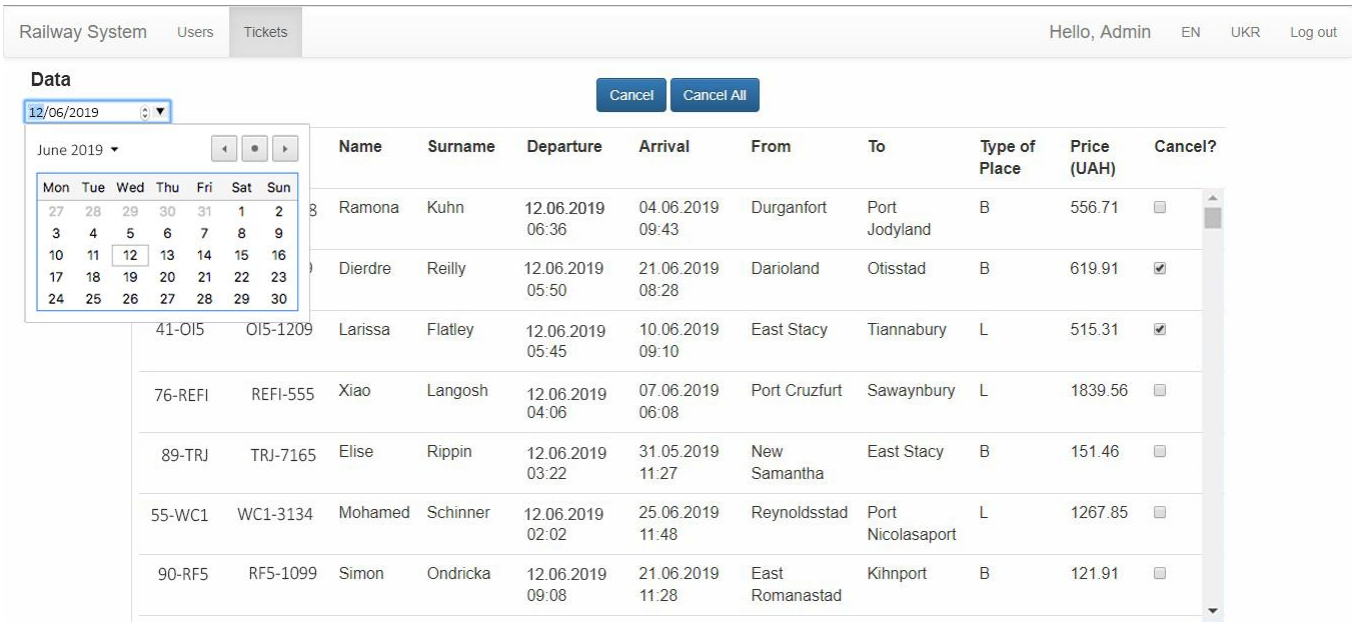

Рисунок 5.12 — Меню управління забронюваними квитками

Висновки до розділу:

В даному розділі описано роботу реалізованої інформаційної клієнт-серверної системи обробки даних. Надано інструкцію роботи з системою зі сторони клієнта, використовуючи різні рівні доступу.

ВИСНОВКИ

В процесі виконання даного дипломного проекту було розглянуто та досліджено процес розробки архітектурного рівня бази даних, ефективної обробки даних та можливість кращого рішення. На основі цього була створена інформаційна клієнт-серверна система обробки даних на основі системи про бронювання залізничних квитків.

- проаналізовано види архітектурних рішень рівня бази даних для ефективної обробки великих даних та проаналізовано автоматизаційний процес бронювання квитків. Сформульована проблематика обробки великих даних. Для вирішення поставленої задачі було вирішено створити клієнт-серверний застосунок з розподіленою нереляційною базою даних, адже завдяки такому підходу можливе підвищення ефективності обробки великих даних.
- поставлено задачу розробки клієнт-серверної масштабованої системи обробки даних про бронювання залізничних квитків з використанням нереляційних розподілених баз даних.
- проведено аналіз стеку технологій для розробки інформаційної масштабованої клієнт-серверної системи обробки даних про бронювання залізничних квитків.
- описано структуру програмної реалізації системи.
- продемонстровано роботу системи за різних рівнів доступу до кожного модуля та надано інструкцію по використанню веб-інтерфейсу.

На основі аналізу програмної реалізації архітектурного рівня бази даних було зроблено висновок, нереляційна розподілена база даних не достатньо практична у використанні для даної інформаційної системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- MONGODB MANUAL [Електронний ресурс] // Fixstarts. – 2008. – Режим доступу до ресурсу: https://docs.mongodb.com/manual/reference/program
- Зайцев П. MySQL и MongoDB — когда и что лучше использовать [Електронний ресурс] / Петр Зайцев. – 24. – Режим доступу до ресурсу: https://habr.com/ru/post/322532/ .
- Zyp K. NoSQL Architecture [Електронний ресурс] / Kris Zyp. – 2017. – Режим доступу до ресурсу: https://www.sitepen.com/blog/nosql-architecture/.
- NoSQL Database Architectural Comparison. // Fixstarts. – 2017. – С. 19.
- Faysal A. Concepts of Database Architecture [Електронний ресурс] / Ahmed Faysal. – 2017. – Режим доступу до ресурсу: https://medium.com/oceanize-geeks/concepts-of-database-architecture-dfdc558a93e4.
- SQL против NoSQL на примере MySQL и MongoDB [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://tproger.ru/translations/sql-vs-nosql/.
- Oluwatosin H. Client-Server Model / Haroon Shakirat Oluwatosin. // IOSR Journal of Computer Engineering (IOSR-JCE). – 2016. – №2278. – С. 61–77.

Додаток 1

Інформаційна масштабована система обробки даних про бронювання (залізничних) квитків в реальному часі на основі нереляційної розподіленої бази даних

Специфікація

УКР.НТУУ“КПІ”.ТР5167_19Б

Аркушів 2

2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТР5167_19Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ».ТР5167_19Б 12-1	Текст програмного модулю	
УКР.НТУУ«КПІ».ТР5167_19Б 13-1	Опис програми	

Додаток 2

Інформаційна масштабована система обробки даних про бронювання (залізничних) квитків в реальному часі на основі нереляційної розподіленої бази даних

Текст програмного модулю

УКР.НТУУ“КПІ”.ТР5167_19Б 12-1

Аркушів 4

2019

Текст набору методів для обміну даними з базою даних MongoDB

```
public List<User> findAll() {
              MongoCollection<Document>                        collection                        =
       MongoDbConnectionPool.getInstance().getConnection()
                       .getCollection(COLLECTION_NAME);
              List<User> users = new ArrayList<>((int) collection.count());

              for (Document document : collection.find()) {
                  users.add(getUser(document));
              }

              LOG.info(LogMessageDAOUtil.createInfoFindAll(COLLECTION_NAME));
              return users;
          }

  public User findById(Long id) {
              MongoCollection<Document>                        collection                        =
       MongoDbConnectionPool.getInstance().getConnection()
                       .getCollection(COLLECTION_NAME);

              Document document = collection.find(eq(LABEL_ID, id)).first();
              return document == null || document.isEmpty() ? null : getUser(document);
          }

 public User findByEmail(String login) {
              MongoCollection<Document>                        collection                        =
       MongoDbConnectionPool.getInstance().getConnection()
                       .getCollection(COLLECTION_NAME);

              Document document = collection.find(eq(LABEL_EMAIL, login)).first();
              return document == null || document.isEmpty() ? null : getUser(document);
          }


public User create(User user) {
              MongoCollection<Document>                        collection                        =
       MongoDbConnectionPool.getInstance().getConnection()
                       .getCollection(COLLECTION_NAME);

              Document document = new Document();
              document.put(LABEL_ID, user.getId());
              document.put(LABEL_EMAIL, user.getEmail());
              document.put(LABEL_PASSWORD, user.getPassword());
              document.put(LABEL_NAME, user.getName());
              document.put(LABEL_SURNAME, user.getSurname());
              document.put(LABEL_PHONE, user.getPhone());
              document.put(LABEL_ADMIN, user.getAdmin());
              collection.insertOne(document);

              LOG.info(LogMessageDAOUtil.createInfoCreate(COLLECTION_NAME, user.getId()));
              return user;
          }

public User update(User user) {
              throw new IllegalStateException("Not implemented yet!");
          }
```

```
public void delete(User user) {
            MongoCollection<Document>                       collection                      =
      MongoDbConnectionPool.getInstance().getConnection()
                    .getCollection(COLLECTION_NAME);

            collection.deleteOne(eq(LABEL_ID, user.getId()));

            LOG.info(LogMessageDAOUtil.createInfoDelete(COLLECTION_NAME, user.getId()));
        }

private User getUser(Document document) {
            User result = new User();

            result.setId(document.get(LABEL_ID, Number.class).longValue());

            result.setEmail(document.getString(LABEL_EMAIL));
            result.setPassword(document.getString(LABEL_PASSWORD));
            result.setName(document.getString(LABEL_NAME));
            result.setSurname(document.getString(LABEL_SURNAME));
            result.setPhone(document.getString(LABEL_PHONE));

            result.setAdmin(document.getBoolean(LABEL_ADMIN));
            return result;
        }
    }
```

Текст набору методів для обміну даними з JSP-сторінками

```
    public void init() throws ServletException {
            super.init();
            Locale.setDefault(Locale.US);
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            processRequest(request, response);
        }

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            processRequest(request, response);
        }

        protected  void  processRequest(HttpServletRequest  request,  HttpServletResponse
    response)
                throws ServletException, IOException {
            String page;
            try {
                HttpSession session = request.getSession(false);
                if (session == null) {
                    request.getSession(true);
                }

                Command command = controllerHelper.getCommand(request);
                page = command.execute(request, response);
            } catch (ServletException e) {
                //LOG.severe(e.getMessage());
```

```
            LOG.log(Level.SEVERE, e.getMessage(), e);
            request.setAttribute(MESSAGE_ERROR_ATTRIBUTE,
Message.getInstance().getMessage(Message.SERVLET_EXCEPTION));
            page = Configuration.getInstance().getConfig(Configuration.ERROR);

        } catch (IOException e) {
            //LOG.severe(e.getMessage());
            LOG.log(Level.SEVERE, e.getMessage(), e);
            request.setAttribute(MESSAGE_ERROR_ATTRIBUTE,
Message.getInstance().getMessage(Message.IO_EXCEPTION));
            page = Configuration.getInstance().getConfig(Configuration.ERROR);

        } catch (Exception e) {
            //LOG.severe(e.getMessage());
            LOG.log(Level.SEVERE, e.getMessage(), e);
            request.setAttribute(MESSAGE_ERROR_ATTRIBUTE,
Message.getInstance().getMessage(Message.EXCEPTION));
            page = Configuration.getInstance().getConfig(Configuration.ERROR);

        }

        if (page == null) {
            LOG.severe(PAGE_IS_NULL);
            request.setAttribute(MESSAGE_ERROR_ATTRIBUTE,
Message.getInstance().getMessage(Message.PAGE_IS_NULL));
            page = Configuration.getInstance().getConfig(Configuration.ERROR);
        }

        response.setCharacterEncoding(CHARACTER_ENCODING);
        response.setContentType(CONTENT_TYPE);
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(page);
        dispatcher.forward(request, response);
    }
```

Додаток 3

Інформаційна масштабована система обробки даних про бронювання (залізничних) квитків в реальному часі на основі нереляційної розподіленої бази даних

Опис програмного модулю

УКР.НТУУ“КПІ”.ТР5167_19Б 13-1

Аркушів 5

2019

АНОТАЦІЯ

Метою роботи було створення масштабованої інформаційної системи обробки даних про бронювання залізничних квитків з використанням нереляційної розподіленої бази даних, як ефективного рішення рівня бази даних. Програма забезпечує ефективну обробку даних заброньованих квитків . Розроблений архітектурний рівень бази даних може бути використаний на підприємствах, які розроблюють систему обробки великих даних.

ЗМІСТ

1 ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

Даний програмний модуль розроблено у середовищі IntelliJ IDEA, , використовуючи типізовану мову програмування Java, мову розмітки HTML з додаванням JSP технології та деякі необхідні додаткові фреймворки.
Програма призначена для ефективної обробки даних

1.1. Опис логічної структури

Було розроблено програмний продукт, основною задачею якого є ефективна обробка даних.
Структура програми містить в собі такі модулі як:

- Controller, який відповідає за приймання та відповідь запитів(get/post), який відбувається за протоколом HTTP;
- Command, в якому відбувається обробка запитів;
- Service — це модуль бізнес-логіки;
- util, dao, dto — модулі, які відповідають за взаємодію з базою даних.

1.2. Архітектурний рівень бази даних

Архітектурний рівень бази даних розроблений з використанням нереляційної розподіленої бази даних MongoDB.

Розроблено горизонтальний підхід масштабування системи. База даних розподілена на 2 кластери, та головного роутер-сервера, який розподіляє дані по вузлам, використовуючи унікальний ключ шардінку.

1.3. Вхідні та вихідні дані

Вхідними даними для системи є :

- адміністратор;
- станції;
- маршрути;
- ціни;
- вільні місця.

Вихідними даними є:

- заброньовані квитки.

2 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано в браузері Chrome 77.0 на персональному комп’ютері, який працює на базі процесору 2,6 GHz Intel Core i7 та має 16 Гб оперативної пам’яті.

Мінімальні характеристики для розгортання даної системи локально мають бути не менше, ніж 2-х ядерний процесор з частотою 2.0 + GHz , та оперативною пам’яттю в 8Гб.